

David Fink (pro hac vice)
7519 Apache Plume
Houston, TX 77071
Tel.: (713) 729-4991
Fax.: (713) 729-4951
texascowboy6@gmail.com

Duncan M. McNeill
1514 Van Dyke Avenue
San Francisco, CA 94124
Tel.: (650) 994-2295
Fax: (650) 994-2297
dmcneill1@netzero.com
Fed. Bar No. 136416

Attorneys for the Plaintiff

UNITED STATES DISTRICT COURT
FOR THE NORTHERN DISTRICT OF CALIFORNIA

FUZZYSHARP TECHNOLOGIES
INCORPORATED,

Plaintiff,

vs.

NVIDIA CORPORATION,

Defendant.

Civil Action No. 3:12-CV-06375 JST

PLAINTIFF FUZZYSHARP
TECHNOLOGIES CORPORATION'S
OPPOSITION NVIDIA
CORPORATION'S MOTION TO
DISMISS

Date: April 18, 2013
Time: 2:00 PM
Court: Courtroom 9, 19th Floor
Judge: Jon S. Tigar

Plaintiff FuzzySharp Technologies Incorporated ("FST") respectfully submits this
Opposition to the Motion to Dismiss submitted by Nvidia Corporation "Nvidia".

PLAINTIFF FUZZYSHARP TECHNOLOGIES CORPORATION'S OPPOSITION NVIDIA CORPORATION'S
MOTION TO DISMISS

FuzzySharp Technologies Incorporated v. Nvidia Corporation

Civil Action No. 3:12-CV-06375 JST

Page 1

ADDRESSING NON-LEGAL ARGUMENTS

Before Nvidia reaches its legal arguments at page 6, Nvidia makes many assertions not as to law, but possibly for supercilious reasons. Nvidia repeatedly dwells on typo errors, but Nvidia does not argue that the typo errors make the Complaint incomprehensible, or that the Complaint should be dismissed based on case law forbidding typo errors in a Complaint. Nvidia also asserts that FST is a non-practicing entity, possibly suggesting that a large company manufacturing, selling, and/or offering an infringing product has more significant standing in law than a company owning patent rights that have been recognized by the largest engineering organization in the world, the Institute of Engineers and Electronic Engineers, as one of the key patents in its field as is the case here. See Ex. A, particularly the top line identifying the parent U. S. Patent No. 5,914,721 of the patents-in-suit with patentee Dr. Hong Lip Lim, the president of FST.

The logical inconsistencies of the Motion are significant. At page 1, second paragraph, Nvidia asks the Court ambiguously to dismiss Plaintiff's claims for two causes: for indirect infringement and willful infringement for both patents; however, Nvidia in its list of issues on the same page raises the stakes and asserts three issues to be dismissed: 1) indirect infringement; 2) willful infringement; and 3) **direct infringement**. In contrast, the Brief at page 11, under the heading, "Conclusions" asks the Court to dismiss only the Plaintiff's claims for indirect and willful infringement, and adds, a request for Count Two to be dismissed. Count One is not mentioned specifically even though it is admittedly the same as Count Two, but for the patent number. In contrast, the "Conclusions" does not mention direct infringement directly.

Nvidia urges the Court that it is incomprehensible for the Plaintiff to amend the Complaint factually to correct any omitted factual information, if the Court deems the Complaint insufficient; however, Nvidia is unable to support such an argument other than the fact that the patents-in-suit have expired. Most important, Nvidia provides no legal basis against litigation on the basis of expired patents.

1 In addition, it is respectfully pointed out that all of the patent claims of the patents-in-suit
2 are process claims, and Nvidia has not argued that process claims require any marking
3 requirements for damages.

4 Nvidia argues that there are no facts available to the Plaintiff because the patents-in-suit
5 have expired. Apparently, Nvidia takes the position that when the patents-in-suit expired, the
6 past activities of Nvidia and related facts also expired.

7 Nvidia repeats several times that in the previous case, Nvidia brought a Motion to
8 Dismiss based on collateral estoppel and FST voluntarily dismissed its case (without prejudice),
9 soon thereafter. See for example, the Brief at page 3, second full paragraph. It is not clear what
10 legal point Nvidia is endeavoring to make, but it is obvious that FST respected the Court and did
11 not want to waste judicial time and effort. If FST had opposed the Nvidia's Motion, then Nvidia
12 would have filed a Reply Brief followed by oral arguments. The best Nvidia could achieve with
13 its Motion would be a dismissal without prejudice. After all, the Judge in the previous Nvidia
14 case knew that the determination of invalidity of the patents was not final. FST dismissed the
15 case without prejudice, thereby saving the Court and everyone else from wasting time and
16 resources. Nvidia did not oppose the Motion for voluntary dismissal without prejudice. Thus,
17 Nvidia acquiesced, thereby accepting the dismissal motion by FST. The Court, of course, made
18 a sound decision, and Nvidia appears to dispute its own conduct in that case.

19 Nvidia argues that there was a two month delay in serving it as though the Fed. R. Civ. P.
20 4 (m) limit of 120 days to serve a Summons and Complaint did not exist. Nvidia fails to mention
21 that it was served a notice of lawsuit, copies of a request for a waiver and a copy of a Complaint
22 which it ignored even after the deadline of 30 days to respond. A Motion for fees and costs to
23 serve Nvidia will be filed in due course. Nvidia also failed to mention that it ignored many
24 telephone calls from the process server for FST for almost one week because the process server
25 wanted to identify the proper person for service as a courtesy.

Legal arguments

In general, Nvidia's Brief appears to be a proforma Brief for defending patent infringement with some small changes to personalize the Brief to this case. The cases cited are standard and the limited descriptions are carefully drafted to avoid actually informing the Court of the respective holdings with respect to details of the Complaint. Hence, much of the Brief is uninformative as to the specific Complaint in this case.

As to direct infringement, Nvidia has repeatedly relied on *Twombly* and its progeny and has mentioned *In re Bill of Lading Transmission & Processing Sys.* In its Brief. Nvidia fails to note that the CAFC sat *en banc* for *In re Bill of Lading Transmission & Processing Sys* stated:

Accordingly, to the extent the parties argue that *Twombly* and its progeny conflict with the Forms and create differing pleading requirements, the Forms control.

The CAFC also points out that Form 18 of the Federal Rules of Civil Procedure requires a minimum number of allegations for direct infringement:

(1) an allegation of jurisdiction; (2) a statement that the plaintiff owns the patent; (3) a statement that defendant has been infringing the patent 'by making, selling, and using [the device] embodying the patent'; (4) a statement that the plaintiff has given the defendant notice of its infringement; and (5) a demand for an injunction and damages.

Nvidia has not disputed that any of the required allegations stated by the CAFC is missing from the Complaint, because none of them are missing except, of course, a demand for injunctive relief which is inappropriate for the facts of the case based on expired patents-in-suit.

Hence, the Complaint meets the requirements for direct infringement. Nvidia is the direct infringer because, as the Complaint states, Nvidia is "selling, and offering to sell Graphic Processors (sic) Units capable of performing occlusion culling during computer graphics rendering process." Nvidia has not disputed that it sells the identified infringing Units. FST owns the patents-in-suit, and Nvidia had prior notice of its infringement.

1 Nvidia takes the position that being “capable” does not mean that the Nvidia product does
2 not “necessarily” perform the infringing process. See Nvidia Brief at page 8, lines 1-3. The
3 conclusion urged by Nvidia is without any logical basis. Whether or not there is infringement is
4 for this Court to determine, not Nvidia. Form 18 requires a statement as to infringement.

5 It is true that Nvidia graphic processors which are capable of performing occlusion
6 culling are used for mundane graphics tasks, but that is not the important use of such units, or
7 substantial non-infringing use.

8 Computer manufacturers include Nvidia graphic products capable of occlusion culling is to
9 enable fast rendering of graphic images. The reason gamers and the like pay big bucks for
10 Nvidia graphic processor units is for the capability of rendering 3D games beautifully using
11 occlusion culling to very quickly create images for display on a computer monitor or the like. It
12 is respectfully noted that Nvidia has arrogantly named the occlusion culling concept invented by
13 the president of FST, Dr. Hong Lip Lim, after itself, thereby denying the significant contribution
14 to the graphics field by Dr. Lim. See Ex. B.

15 **As to willful infringement**, Nvidia in its Brief at the top of page 10 states that Nvidia
16 had no reason to believe that the patents were valid and infringed because Plaintiff did not
17 reinitiate its dismissed lawsuit against Nvidia and settled with 3DLabs before the issue of
18 invalidity could be resolved. This argument is actually an admission against interest by Nvidia.
19 First, Nvidia does not state that it was relying on a legal opinion. While the management at
20 Nvidia maybe very smart, a business decision to ignore the patents-in-suit does not avoid willful
21 infringement, particularly if the management had notice of the patents-in-suit from prior
22 litigation. It is a business risk of willful infringement, that Nvidia thought it could do in defiance
23 of FST’s patent rights. The second reason Nvidia argues is that the validity of the patents-in-suit
24 were not established. It is fundamental law that a patent is presumed to be valid, and Nvidia
25 would have learned this if it consulted a competent law firm. Thus, Nvidia has effectively argued
26 that Nvidia management decided to ignore the notice of the infringement of the patents-in-suit,
27 and proceeded to infringe the patents-in-suit willfully by selling and offering to sell infringing

1 Units.

2 **As to indirect infringement**, the Nvidia Brief at page 8 argues in the first full paragraph:
3 “NVIDIA could not induce infringement of patents it reasonably believed were invalid.” Once
4 again, Nvidia relies on its management’s business decision, not a legal opinion, that the patents-
5 in-suit were invalid. Nvidia argues that there is no indirect infringement based on the speculation
6 of Nvidia’s management that the patents-in-suit were invalid.

7 It is true that Judge Armstrong dismissed the case against 3DLabs on the basis that the
8 claims asserted in that case were invalid; however, an appeal was taken so the issue of validity
9 remained incomplete despite the well reasoned Decision. The possibility that the CAFC might
10 find a different understanding exists. Hence, even if Nvidia suddenly found it had a legal
11 opinion in hand at that time, the subsequent action by the CAFC would certain require a revision
12 of such a legal decision after the CAFC made its decision. Thus, as a defense to the Complaint,
13 Nvidia would need to suddenly “find” two legal opinions, but such legal opinions would not
14 provide a basis for dismissing the Complaint as to indirect infringement. At best, such opinions
15 would only be useful as a defense in an Answer, not a basis for dismissal.

16 **As to contributory infringement**, it is respectfully pointed out that Nvidia sells and
17 offers its Graphic Processor Units capable of performing occlusion culling during the computer
18 graphics rendering process. This is set forth at Count One, para 13, and Count Two, para. 16 of
19 the Complaint. Nvidia had notice from the previous lawsuit by FST of infringing the patents-in-
20 suit. Hence, it is it is ingenuous for Nvidia to assert at page 9 of its Brief that it did not believe
21 that its product was sold for the purpose of enabling patent infringement. Nvidia has not denied
22 that it sells the identified infringing products for enabling users to infringe the patents-in-suit.
23 Hence, Nvidia is a contributory infringer.

1 **CONCLUSION**

2 Defendant's arguments as to the issues of direct infringement, wilful infringement, and
3 indirect infringement, inducement and contributory infringement, have been shown to have no
4 basis in law. In addition, if the Court deems that an amended Complaint might clarify some
5 issues, the Plaintiff is prepared to comply.

6 THE PLAINTIFF
7 FUZZYSHARP TECHNOLOGIES
8 INCORPORATED

9 Date: March 14, 2013

10 By: /S/ David Fink
11 David Fink
12 Attorney for Plaintiff
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

28

PLAINTIFF FUZZYSHARP TECHNOLOGIES CORPORATION'S OPPOSITION NVIDIA CORPORATION'S
MOTION TO DISMISS

FuzzySharp Technologies Incorporated v. Nvidia Corporation

Civil Action No. 3:12-CV-06375 JST

Page 7

EX. A



The Influence of IEEE on Key Patents

Prepared For:

IEEE

445 Hoes Lane

P.O. Box 1331

Piscataway, NJ 08855-1331, USA

Prepared By:

Patrick Thomas, PhD

and

Anthony Breitzman, PhD

1790 Analytics, LLC

East Gate Center, Suite 200

309 Fellowship Road

Mount Laurel, NJ 08054

January 12, 2006

Computer Software

| <i>Patent Title</i> | <i>Citation Index</i> | <i>Cite Count</i> | <i>Pub Year</i> | <i>inventors</i> | <i>Assignee</i> | <i>Parent</i> | <i>Ult Parent</i> |
|---------------------|-----------------------|-------------------|-----------------|------------------|-----------------|---------------|-------------------|
| 05914721 | 4.54226 | 38 | 1999 | Lim; Hong Lip | UNASSIGNED | | |

Visibility calculations for 3D computer graphics

NPR Text

- 74 An optimal algorithm for detecting weak visibility of a polygon J. Sack S. Suri IEEE transactions on computers, vol. 39, No. 10, Oct. 1990.
- 38 Radiosity redistribution for dynamic environment D. George F. Sillion D. Greenberg IEEE Computer Graphics & Applications, vol. 4, 1990.
- 179 Z. Gigus et al., "Computing the aspect graph for line drawings of polyhedral objects", in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Computer Society Press, New York, pp. 654-661, 1988.
- 171 S. Chang, "On fuzzy mapping and control", IEEE Transactions on Systems, Man & Cybernetics, vol. SMC-2, No. 1, pp. 30-34, 1972.
- 150 C. Hornung, "A method for solving the visibility problem", IEEE Computer Graphics & Applications, vol. 4, pp. 26-33, 1984.
- 143 E. Catmull, "Computer display of curved surfaces" in IEEE Transactions of Computers, pp. 309-315, 1971.
- 132 D. Gordon et al., "Front-to-back display of BSP trees", IEEE Computer Graphics & Applications, vol. 11, pp. 79-85, 1991.
- 121 Spatial transformation for rapid scan-line surface shadowing P. Robertson IEEE computer graphics & applications Mar. 1989.
- 52 Generating soft shadows with a depth buffer algorithm Brotman L. S. IEEE Computer Graphics and Applications vol. 4, No. 10, 1984.
- 51 Me and my (fake shadow) Blinn, J. F. IEEE Computer Graphics and Applications, vol. 8, No. 1 1988.
- 50 A general version of crow's shadow volumes Bergeron P. IEEE Computer Graphics and Applications, vol. 6, No. 9, Sep. 1986.
- 24 A solution to the hidden-line problem for computer-drawn polyhedra P. Loutrel IEEE Transactions on Computers, Mar. 1970.
- 39 A survey of shadow algorithms A. Woo, F. Poulin A. Fournier IEEE Computer Graphics & Applications, vol. 6, 1990.
- 47 Realism in computer graphics: a survey. J. Amanatides IEEE Computer Graphics and Applications, vol. 7, No. 1, 1987.
- 59 The light buffer: a shadow-testing accelerator Haines, E. A. Greenberg, D. IEEE Computer Graphics and Applications, vol. 6, No. 9, Sep. 1986.

05918225 4.48085 78 1999 White; Peter W. French; Clark D. Chen;
Yong Min Yach; David

SQL-based database system with improved indexing methodology

NPR Text

- 8 Graefe et al., "Data Compression and Database Performance," IEEE, Applied Computing Symposium, 1991, pp. 22-27.
- 9 Perrizo et al., "Domain Vector Accelerator (DVA): A Query Accelerator for Relational Operations," IBM Corp., Rochester, MN, IEEE, Data Engineering, 7th Annual International Conference, 1991, pp. 491-498.
- 5 Chu et al., "A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems," IEEE, v19, n8, IEEE Transactions on Software Engineering, Aug. 1993, pp. 804-812.
- 2 Reinartz, K., "Aspects of vertical mode in multiprocessor systems, unconventional computation on conventional processors," Second International Specialist Seminar on the Design and Application of Parallel Digital Processors, IEEE, 1991, pp. 48-54.

EX. B

Name

NV_occlusion_query

Name Strings

GL_NV_occlusion_query

Contact

Matt Craighead, NVIDIA Corporation (mcraighead 'at' nvidia.com)

Notice

Copyright NVIDIA Corporation, 2001, 2002.

IP Status

NVIDIA Proprietary.

Status

Shipping (version 1.0)

Version

NVIDIA Date: February 6, 2002 (version 1.0)

Number

261

Dependencies

Written based on the wording of the OpenGL 1.3 specification.

Requires support for the HP_occlusion_test extension.

Overview

The HP occlusion_test extension defines a mechanism whereby an application can query the visibility of an object, where "visible" means that at least one pixel passes the depth and stencil tests.

The HP extension has two major shortcomings.

- It returns the result as a simple GL_TRUE/GL_FALSE result, when in fact it is often useful to know exactly how many pixels passed.
- It provides only a simple "stop-and-wait" model for using multiple queries. The application begins an occlusion test and ends it; then, at some later point, it asks for the result, at which point the driver must stop and wait until the result from the previous test is back before the application can even begin the next one. This is a very simple model, but its performance is mediocre when an application wishes to perform many queries, and it eliminates most of the opportunities for parallelism between the CPU and GPU.

This extension solves both of those problems. It returns as its result the number of pixels that pass, and it provides an interface conceptually similar to that of NV_fence that allows applications to issue many occlusion queries before asking for the result of any one. As a result, they can overlap the time it takes for the occlusion query results to be returned with other, more useful work, such as rendering other parts of the scene or performing other computations on the CPU.

There are many situations where a pixel count, rather than a boolean result, is useful.

- If the visibility test is an object bounding box being used to decide whether to skip the object, sometimes it can be acceptable, and beneficial to performance, to skip an object if less than some threshold number of pixels could be visible.
- Knowing the number of pixels visible in the bounding box may also help decide what level of detail a model should be drawn with. If only a few pixels are visible, a low-detail model may be acceptable. In general, this allows level-of-detail mechanisms to be slightly less ad hoc.
- "Depth peeling" techniques, such as order-independent transparency, would typically like to know when to stop rendering more layers; it

is difficult to come up with a way to determine a priori how many layers to use. A boolean count allows applications to stop when more layers will not affect the image at all, but this will likely be unacceptable for performance, with minimal gains to image quality. Instead, it makes more sense to stop rendering when the number of pixels goes below a threshold; this should provide better results than any of these other algorithms.

- Occlusion queries can be used as a replacement for `glReadPixels` of the depth buffer to determine whether, say, a light source is visible for the purposes of a lens flare effect or a halo to simulate glare. Pixel counts allow you to compute the percentage of the light source that is visible, and the brightness of these effects can be modulated accordingly.

Issues

- * Should we use an object-based interface?

RESOLVED: Yes, this makes the interface much simpler, and it is friendly for indirect rendering.

- * Should we offer an entry point analogous to `glTestFenceNV`?

RESOLVED: No, it is sufficient to have `glGetOcclusionQueryivNV` provide a query for whether the occlusion query result is back yet. Whereas it is interesting to poll fence objects, it is relatively less interesting to poll occlusion queries.

- * Is `glGetOcclusionQueryiivNV` necessary?

RESOLVED: Yes, it makes using a 32-bit pixel count less painful.

- * Should there be a limit on how many queries can be outstanding?

RESOLVED: No. This would make the extension much more difficult to spec and use. Allowing this does not add any significant implementation burden; and even if drivers have some internal limit on the number of outstanding queries, it is not expected that applications will need to know this to achieve optimal or near-optimal performance.

- * What happens if `glBeginOcclusionQueryNV` is called when an occlusion query is already outstanding for a different object?

RESOLVED: This is a `GL_INVALID_OPERATION` error.

- * What happens if `HP_occlusion_test` and `NV_occlusion_query` usage is overlapped?

RESOLVED: The two can be overlapped safely. Counting is enabled if we are either inside a `glBeginOcclusionQueryNV` or if `GL_OCCLUSION_TEST_HP` is enabled. The alternative (producing an error) does not work -- it would require that `glPopAttrib` be capable of producing an error, which would be rather problematic.

Note that `glBeginOcclusionQueryNV`, not `glEndOcclusionQueryNV`, resets the pixel counter and occlusion test result. This can avoid certain types of strange behavior where an occlusion query's pixel count does not always correspond to the pixels rendered during the occlusion query. The spec would make sense the other way, but the behavior would be strange.

- * Does `EndOcclusionQuery` need to take any parameters?

RESOLVED: No. Giving it, for example, an "id" parameter would be redundant -- adding complexity for no benefit. Only one query can be active at a time.

- * How many bits should we require the pixel counter to be, at minimum?

RESOLVED: 24. 24 is enough to handle 8.7 full overdraws of a 1600x1200 window. That seems quite sufficient.

- * What should we do about overflows?

RESOLVED: Overflows leave the pixel count undefined. Saturating is recommended but not required.

The ideal behavior really is to saturate. This ensures that you always get a "large" result when you render many pixels. It also

ensures that apps which want a boolean test can do one on their own, and not worry about the rare case where the result ends up exactly at zero from wrapping.

That being said, with 24 bits of pixel count required, it's not clear that this really matters. It's better to be a bit permissive here. In addition, even if saturation was required, the goal of having strictly defined behavior is still not really met.

Applications don't (or at least shouldn't) check for some exact number of bits. Imagine if a multitextured app had been written that required that the number of texture units supported be exactly two! Implementors of OpenGL would be greatly annoyed to find that the app did not run on, say, three-texture or four-texture hardware.

So, we expect apps here to always be doing a "greater than or equal to" check. An app might check for, say, at least 28 bits. This doesn't ensure defined behavior -- it only ensures that once an overflow occurs (which may happen at any power of two), that overflow will be handled with saturation. This behavior still remains sufficiently unpredictable that the reasons for defining behavior in even rarely-used cases (preventing compatibility problems, for example) are unsatisfied.

All that having been said, saturation is still explicitly recommended in the spec language.

- * What is the interaction with multisample, which was not defined in the original spec?

RESOLVED: The pixel count is the number of samples that pass, not the number of pixels. This is true even if GL_MULTISAMPLE is disabled but GL_SAMPLE_BUFFERS is 1. Note that the depth/stencil test optimization whereby implementations may choose to depth test at only one of the samples when GL_MULTISAMPLE is disabled does not cause this to become ill-specified, because we are counting the number of samples that are still alive after the depth test stage. The mechanism used to decide whether to kill or keep those samples is not relevant.

- * Exactly what stage are we counting at? The original spec said depth test; what does stencil test do?

RESOLVED: We are counting immediately after both the depth and stencil tests, i.e., pixels that pass both. This was the original spec's intent. Note that the depth test comes after the stencil test, so to say that it is the number that pass the depth test is reasonable; though it is often helpful to think of the depth and stencil tests as being combined, because the depth test result impacts the stencil operation used.

- * Is it guaranteed that occlusion queries return in order?

RESOLVED: Yes. It makes sense to do this. If occlusion test X occurred before occlusion query Y, and the driver informs the app that occlusion query Y is done, the app can infer that occlusion query X is also done. For applications that do poll, this allows them to do so with less effort.

- * Will polling an occlusion query without a glFlush possibly cause an infinite loop?

RESOLVED: Yes, this is a risk. If you ask for the result, however, any flush required will be done automatically. It is only when you are polling that this is a problem because there is no guarantee that a flush has occurred in the time since glEndOcclusionQueryNV, and the spec is written to say that the result is only "available" if the value could be returned instantaneously.

This is different from NV_fence, where FinishFenceNV can cause an app hang, and where TestFenceNV was also not guaranteed to ever finish.

There need not be any spec language to describe this behavior because it is implied by what is already said.

In short, if you use GL_PIXEL_COUNT_AVAILABLE_NV, you must use glFlush, or your app may hang.

- * The HP occlusion test specs did not contain the spec edits that explain the exact way the extension works. Should this spec fill in those details?

RESOLVED: Yes. These two extensions are intertwined in so many important ways that doing so is not optional.

- * Should there be a "target" parameter to BeginOcclusionQuery?

RESOLVED: No. We're not trying to solve the problem of "query anything" here.

- * What might an application that uses this extension look like?

Here is some rough sample code:

```
GLuint occlusionQueries[N];
GLuint pixelCount;

glGenOcclusionQueriesNV(N, occlusionQueries);
...
// before this point, render major occluders
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glDepthMask(GL_FALSE);
// also disable texturing and any fancy shading features
for (i = 0; i < N; i++) {
    glBeginOcclusionQueryNV(occlusionQueries[i]);
    // render bounding box for object i
    glEndOcclusionQueryNV();
}
// at this point, if possible, go and do some other computation
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
glDepthMask(GL_TRUE);
// reenable other state
for (i = 0; i < N; i++) {
    glGetOcclusionQueryuivNV(occlusionQueries[i], GL_PIXEL_COUNT_NV,
                           &pixelCount);
    if (pixelCount > 0) {
        // render object i
    }
}
```

- * Is this extension useful for saving geometry, fill rate, or both?

It is expected that it will be most useful for saving geometry work, because for the cost of rendering a bounding box you can save rendering a normal object.

It is possible for this extension to help in fill-limited situations, but using it may also hurt performance in such situations, because rendering the pixels of a bounding box is hardly free. In most situations a bounding box will probably have more pixels than the original object.

One exception is that for objects rendered with multiple passes, the first pass can be wrapped with an occlusion query almost for free. That is, render the first pass for all objects in the scene, and get the number of pixels rendered on each object. If zero pixels were rendered for an object, you can skip subsequent rendering passes. This trick can be very useful in many cases.

- * What can be said about guaranteeing correctness when using occlusion queries, especially as it relates to invariance?

Invariance is critical to guarantee the correctness of occlusion queries. If occlusion queries go through a different code path than standard rendering, the pixels rendered may be different.

However, the invariance issues are difficult at best to solve. Because of the vagaries of floating-point precision, it is difficult to guarantee that rendering a bounding box will render at least as many pixels with equal or smaller Z values than the object itself would have rendered.

Likewise, many other aspects of rendering state tend to be different when performing an occlusion query. Color and depth writes are typically disabled, as are texturing, vertex programs, and any fancy per-pixel math. So unless all these features have guarantees of invariance themselves (unlikely at best), requiring invariance for NV occlusion query would be futile.

For what it's worth, NVIDIA's implementation is fully invariant with respect to whether an occlusion query is active; that is, it does not affect the operation of any other stage of the pipeline. (When occlusion queries are being emulated on hardware that does not support them, via the emulation registry keys, using an occlusion query produces a software rasterization fallback, and in such cases invariance cannot be guaranteed.)

Another problem that can threaten correctness is near and far clipping. If the bounding box penetrates the near clip plane, for example, it may be clipped away, reducing the number of pixels counted, when in fact the original object may have stayed entirely beyond the near clip plane. Whenever you design an algorithm using occlusion queries, it is best to be careful about the near and far clip planes.

- * How can frame-to-frame coherency help applications using this extension get even higher performance?

Usually, if an object is visible one frame, it will be visible the next frame, and if it is not visible, it will not be visible the next frame.

Of course, for most applications, "usually" isn't good enough. It is undesirable, but acceptable, to render an object that wasn't visible, because that only costs performance. It is generally unacceptable to not render an object that was visible.

The simplest approach is that visible objects should be checked every N frames (where, say, N=5) to see if they have become occluded, while objects that were occluded last frame must be rechecked again in the current frame to guarantee that they are still occluded. This will reduce the number of wasteful occlusion queries by a factor of almost N.

It may also pay to do a raycast on the CPU in order to try to prove that an object is visible. After all, occlusion queries are only one of many items in your bag of tricks to decide whether objects are visible or invisible. They are not an excuse to skip frustum culling, or precomputing visibility using portals for static environments, or other standard visibility techniques.

In general, though, taking advantage of frame-to-frame coherency in your occlusion query code is absolutely essential to getting the best possible performance.

New Procedures and Functions

```
void GenOcclusionQueriesNV(sizei n, uint *ids);
void DeleteOcclusionQueriesNV(sizei n, const uint *ids);
boolean IsOcclusionQueryNV(uint id);
void BeginOcclusionQueryNV(uint id);
void EndOcclusionQueryNV(void);
void GetOcclusionQueryivNV(uint id, enum pname, int *params);
void GetOcclusionQueryiivNV(uint id, enum pname, uint *params);
```

New Tokens

Accepted by the <cap> parameter of Enable, Disable, and IsEnabled, and by the <pname> parameter of GetBooleanv, GetIntegerv, GetFloatv, and GetDoublev:

| | |
|-------------------|--------|
| OCCLUSION_TEST_HP | 0x8165 |
|-------------------|--------|

Accepted by the <pname> parameter of GetBooleanv, GetIntegerv, GetFloatv, and GetDoublev:

| | |
|-------------------------------|--------|
| OCCLUSION_TEST_RESULT_HP | 0x8166 |
| PIXEL_COUNTER_BITS_NV | 0x8864 |
| CURRENT_OCCLUSION_QUERY_ID_NV | 0x8865 |

Accepted by the <pname> parameter of GetOcclusionQueryivNV and GetOcclusionQueryiivNV:

| | |
|--------------------------|--------|
| PIXEL_COUNT_NV | 0x8866 |
| PIXEL_COUNT_AVAILABLE_NV | 0x8867 |

Additions to Chapter 2 of the OpenGL 1.3 Specification (OpenGL Operation)

None.

Additions to Chapter 3 of the OpenGL 1.3 Specification (Rasterization)

None.

Additions to Chapter 4 of the OpenGL 1.3 Specification (Per-Fragment Operations and the Frame Buffer)

Add a new section "Occlusion Tests and Queries" between sections 4.1.6 and 4.1.7:

"4.1.6A Occlusion Tests and Queries

Occlusion testing keeps track of whether any pixels have passed the depth test. Such testing is enabled or disabled with the generic Enable and Disable commands using the symbolic constant `OCCLUSION_TEST_HP`. The occlusion test result is initially `FALSE`.

Occlusion queries can be used to track the exact number of fragments that pass the depth test. Occlusion queries are associated with occlusion query objects. The command

```
void GenOcclusionQueriesNV(sizei n, uint *ids);
```

returns `n` previously unused occlusion query names in `ids`. These names are marked as used, but no object is associated with them until the first time `BeginOcclusionQueryNV` is called on them. Occlusion queries contain one piece of state, a pixel count result. This pixel count result is initialized to zero when the object is created.

Occlusion queries are deleted by calling

```
void DeleteOcclusionQueriesNV(sizei n, const uint *ids);
```

`ids` contains `n` names of occlusion queries to be deleted. After an occlusion query is deleted, its name is again unused. Unused names in `ids` are silently ignored.

An occlusion query can be started and finished by calling

```
void BeginOcclusionQueryNV(uint id);
void EndOcclusionQueryNV(void);
```

If `BeginOcclusionQueryNV` is called with an unused `id`, that `id` is marked as used and associated with a new occlusion query object. If it is called while another occlusion query is active, an `INVALID_OPERATION` error is generated. If `EndOcclusionQueryNV` is called while no occlusion query is active, an `INVALID_OPERATION` error is generated. Calling either `GenOcclusionQueriesNV` or `DeleteOcclusionQueriesNV` while an occlusion query is active causes an `INVALID_OPERATION` error to be generated.

When `EndOcclusionQueryNV` is called, the current pixel counter is copied into the active occlusion query object's pixel count result. `BeginOcclusionQueryNV` resets the pixel counter to zero and the occlusion test result to `FALSE`.

Whenever a fragment reaches this stage and `OCCLUSION_TEST_HP` is enabled or an occlusion query is active, the occlusion test result is set to `TRUE` and the pixel counter is incremented. If the value of `SAMPLE_BUFFERS` is 1, then the pixel counter is incremented by the number of samples whose coverage bit is set; otherwise, it is always incremented by one. If it the pixel counter overflows, i.e., exceeds the value `2^PIXEL_COUNTER_BITS_NV-1`, its value becomes undefined. It is recommended, but not required, that implementations handle this overflow case by saturating at `2^PIXEL_COUNTER_BITS_NV-1` and incrementing no further.

The necessary state is a single bit indicating whether the occlusion test is enabled, a single bit indicating whether an occlusion query is active, the identifier of the currently active occlusion query, a counter of no smaller than 24 bits keeping track of the pixel count, and a single bit indicating the occlusion test result."

Additions to Chapter 5 of the OpenGL 1.3 Specification (Special Functions)

Add to the end of Section 5.4 "Display Lists":

"DeleteOcclusionQueriesNV, GenOcclusionQueriesNV, IsOcclusionQueryNV, GetOcclusionQueryivNV, and GetOcclusionQueryiivNV are not compiled into display lists but are executed immediately."

Additions to Chapter 6 of the OpenGL 1.3 Specification (State and State Requests)

Add a new section 6.1.13 "Occlusion Test and Occlusion Queries":

"The occlusion test result can be queried using GetBooleanv, GetIntegerv, GetFloatv, or GetDoublev with a <pname> of OCCLUSION_TEST_RESULT_HP. Whenever such a query is performed, the occlusion_test_result is reset to FALSE and the pixel counter is reset to zero as a side effect.

Which occlusion query is active can be queried using GetBooleanv, GetIntegerv, GetFloatv, or GetDoublev with a <pname> of CURRENT_OCCLUSION_QUERY_ID_NV. This query returns the name of the currently active occlusion query if one is active, and zero otherwise.

The state of an occlusion query can be queried with the commands

```
void GetOcclusionQueryivNV(uint id, enum pname, int *params);
void GetOcclusionQueryuivNV(uint id, enum pname, uint *params);
```

If the occlusion query object named by id is currently active, then an INVALID_OPERATION error is generated.

If <pname> is PIXEL_COUNT_NV, then the occlusion query's pixel count result is placed in params.

Often, occlusion query results will be returned asynchronously with respect to the host processor's operation. As a result, sometimes, if a pixel count is queried, the host must wait until the result is back. If <pname> is PIXEL_COUNT_AVAILABLE_NV, the value placed in params indicates whether or not such a wait would occur if the pixel count for that occlusion query were to be queried presently. A result of TRUE means no wait would be required; a result of FALSE means that some wait would occur. The length of this wait is potentially unbounded. It must always be true that if the result for one occlusion query is available, the result for all previous occlusion queries must also be available at that point in time."

GLX Protocol

Seven new GL commands are added.

The following two rendering commands are sent to the server as part of a glXRender request:

| | | |
|-----------------------|--------|--------------------------|
| BeginOcclusionQueryNV | | |
| 2 | 8 | rendering command length |
| 2 | ???? | rendering command opcode |
| 4 | CARD32 | id |
| EndOcclusionQueryNV | | |
| 2 | 4 | rendering command length |
| 2 | ???? | rendering command opcode |

The remaining five commands are non-rendering commands. These commands are sent separately (i.e., not as part of a glXRender or glXRenderLarge request), using the glXVendorPrivateWithReply request:

| | | |
|--------------------------|-----------------|----------------------------------------|
| DeleteOcclusionQueriesNV | | |
| 1 | CARD8 | opcode (X assigned) |
| 1 | 17 | GLX opcode (glXVendorPrivateWithReply) |
| 2 | 4+n | request length |
| 4 | ???? | vendor specific opcode |
| 4 | GLX_CONTEXT_TAG | context tag |
| 4 | INT32 | n |
| n*4 | LISTofCARD32 | ids |
| GenOcclusionQueriesNV | | |
| 1 | CARD8 | opcode (X assigned) |
| 1 | 17 | GLX opcode (glXVendorPrivateWithReply) |
| 2 | 4 | request length |
| 4 | ???? | vendor specific opcode |
| 4 | GLX_CONTEXT_TAG | context tag |
| 4 | INT32 | n |
| => | | |
| 1 | 1 | reply |
| 1 | | unused |

```

2          CARD16          sequence number
4          n               reply length
24         unused
n*4        LISTofCARD322  queries

IsOcclusionQueryNV
1          CARD8          opcode (X assigned)
1          17            GLX opcode (glXVendorPrivateWithReply)
2          4             request length
4          ????          vendor specific opcode
4          GLX_CONTEXT_TAG context tag
4          CARD32        id
=>
1          1             reply
1          unused
2          CARD16        sequence number
4          0             reply length
4          BOOL32        return value
20         unused
1          1             reply

GetOcclusionQueryivNV
1          CARD8          opcode (X assigned)
1          17            GLX opcode (glXVendorPrivateWithReply)
2          5             request length
4          ????          vendor specific opcode
4          GLX_CONTEXT_TAG context tag
4          CARD32        id
4          ENUM          pname
=>
1          1             reply
1          unused
2          CARD16        sequence number
4          m             reply length, m=(n==1?0:n)
4          unused
4          CARD32        n

if (n=1) this follows:
4          INT32         params
12         unused

otherwise this follows:
16         unused
n*4        LISTofINT32  params

GetOcclusionQueryiivNV
1          CARD8          opcode (X assigned)
1          17            GLX opcode (glXVendorPrivateWithReply)
2          5             request length
4          ????          vendor specific opcode
4          GLX_CONTEXT_TAG context tag
4          CARD32        id
4          ENUM          pname
=>
1          1             reply
1          unused
2          CARD16        sequence number
4          m             reply length, m=(n==1?0:n)
4          unused
4          CARD32        n

if (n=1) this follows:
4          CARD32        params
12         unused

otherwise this follows:
16         unused
n*4        LISTofCARD32  params

```

Errors

The error INVALID_VALUE is generated if GenOcclusionQueriesNV is called where n is negative.

The error INVALID_VALUE is generated if DeleteOcclusionQueriesNV is called where n is negative.

The error INVALID_OPERATION is generated if GenOcclusionQueriesNV or DeleteOcclusionQueriesNV is called when an occlusion query is active.

The error INVALID_OPERATION is generated if BeginOcclusionQueryNV is called when an occlusion query is already active.

The error INVALID_OPERATION is generated if EndOcclusionQueryNV is called when an occlusion query is not active.

The error INVALID_OPERATION is generated if GetOcclusionQueryivNV or GetOcclusionQueryiivNV is called where id is not the name of an occlusion query.

The error INVALID_OPERATION is generated if GetOcclusionQueryivNV or GetOcclusionQueryiivNV is called where id is the name of the currently active occlusion query.

The error INVALID_ENUM is generated if GetOcclusionQueryivNV or GetOcclusionQueryiivNV is called where pname is not either PIXEL_COUNT_NV or PIXEL_COUNT_AVAILABLE_NV.

The error INVALID_OPERATION is generated if any of the commands defined in this extension is executed between the execution of Begin and the corresponding execution of End.

New State

(table 6.18, p. 226)

| Sec | Get Value Attribute | Type | Get Command | Initial Value | Description |
|--------|-------------------------------|------|-------------|---------------|------------------------|
| | ----- | ---- | ----- | ----- | ----- |
| | OCCLUSION_TEST_HP | B | IsEnabled | FALSE | occlusion test enable |
| 4.1.6A | enable | | | | |
| | OCCLUSION_TEST_RESULT_HP | B | GetBooleanv | FALSE | occlusion test result |
| 4.1.6A | - | | | | |
| | - | B | GetBooleanv | FALSE | occlusion query active |
| 4.1.6A | - | | | | |
| | CURRENT_OCCLUSION_QUERY_ID_NV | Z+ | GetIntegerv | 0 | occlusion query ID |
| 4.1.6A | - | | | | |
| | - | Z+ | - | 0 | pixel counter |
| 4.1.6A | - | | | | |

New Implementation Dependent State

(table 6.29, p. 237) Add the following entry:

| Get Value Attribute | Type | Get Command | Minimum Value | Description | Sec |
|------------------------|------|-------------|---------------|-------------------------------------|--------|
| ----- | ---- | ----- | ----- | ----- | ----- |
| PIXEL_COUNTER_BITS_NV | Z+ | GetIntegerv | 24 | Number of bits in pixel counters | 6.1.13 |

Revision History

none yet